

Recursion: Python

Cheatsheets / Learn Recursion with Python

TOPICS Recursion: Conceptual Recursion: Python

Stack Overflow Error in Recursive Function

A recursive function that is called with an input that requires too many iterations will cause the call stack to get too large, resulting in a stack overflow error. In these cases, it is more appropriate to use an iterative solution. A recursive solution is only suited for a problem that does not exceed a certain number of recursive calls.

For example, myfunction() below throws a stack overflow error when an input of 1000 is used.

def myfunction(n): if n == **0**: return n else: return myfunction(n-1) myfunction(1000) #results in stack overflow error

Fibonacci Sequence

A Fibonacci sequence is a mathematical series of numbers such that each number is the sum of the two preceding numbers, starting from 0 and 1.

Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Call Stack Construction in While Loop

A call stack with execution contexts can be constructed using a while loop, a list to represent the call stack and a dictionary to represent the execution contexts. This is useful to mimic the role of a call stack inside a recursive function.

Binary Search Tree

data structure that makes sorted lists easier to search. Binary search trees: Reference two children at most per tree

In Python, a binary search tree is a recursive

- node. The "left" child of the tree must contain a
- value lesser than its parent.
- The "right" child of the tree must contain a
- value greater than it's parent.
- **Recursion and Nested Lists**



using a recursive function. The base case evaluates an element in the list. If it is not

A nested list can be traversed and flattened

another list, the single element is appended to a flat list. The recursive step calls the recursive function with the nested list element as input.



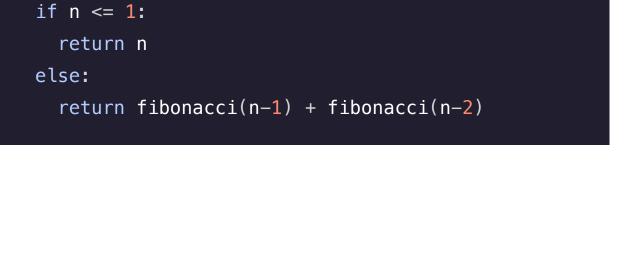
be implemented using recursion. Given an input

Fibonacci Recursion

of index N, the recursive function has two base cases – when the index is zero or 1. The recursive function returns the sum of the index minus 1 and the index minus 2. The Big-O runtime of the Fibonacci function is $O(2^N)$.

Computing the value of a Fibonacci number can

Modeling Recursion as Call Stack



def fibonacci(n):

def countdown(value):

def countdown(value):

print("done")

def build_bst(my_list):

if len(my_list) == 0:

return "No Child"

def depth(tree):

if not tree:

return 0

return n

last_digit = n % 10

def is_palindrome(str):

if str[0] != str[-1]:

return is_palindrome(str[1:-1])

if len(str) < 2:</pre>

return True

return False

fiblist = [0, 1]

return fiblist[n]

for i in range(2,n+1):

sum_digits(552) #returns 12

middle_index = len(my_list) // 2

else:

if value <= 0: #base case</pre>

One can model recursion as a call stack with execution contexts using a while loop and a

Python list. When the base case is reached, print out the call stack list in a LIFO (last in first out) manner until the call stack is empty. Using another while loop, iterate through the call stack list. Pop the last item off the list and add it to a variable to store the accumulative

result. Print the result.

```
call_stack = []
 while value > 0:
   call_stack.append({"input":value})
    print("Call Stack:",call_stack)
    value -= 1
 print("Base Case Reached")
 while len(call_stack) != 0:
    print("Popping {} from call stack".format(call_st
    print("Call Stack:",call_stack)
countdown(4)
Call Stack: [{'input': 4}]
Call Stack: [{'input': 4}, {'input': 3}]
Call Stack: [{'input': 4}, {'input': 3}, {'input': 2}
Call Stack: [{'input': 4}, {'input': 3}, {'input': 2}
Base Case Reached
Popping {'input': 1} from call stack
Call Stack: [{'input': 4}, {'input': 3}, {'input': 2}
Popping {'input': 2} from call stack
Call Stack: [{'input': 4}, {'input': 3}]
Popping {'input': 3} from call stack
Call Stack: [{'input': 4}]
Popping {'input': 4} from call stack
Call Stack: []
\mathbf{I}_{-}\mathbf{I}_{-}\mathbf{I}_{-}
```

In Python, a recursive function accepts an argument and includes a condition to check whether it matches the base case. A recursive

Recursion in Python

function has: Base Case - a condition that evaluates the current input to stop the recursion from continuing.

Recursive Step - one or more calls to the

- recursive function to bring the input closer to the base case.
- **Build a Binary Search Tree**

print(value) countdown(value-1) #recursive case

algorithm do the following: BASE CASE:

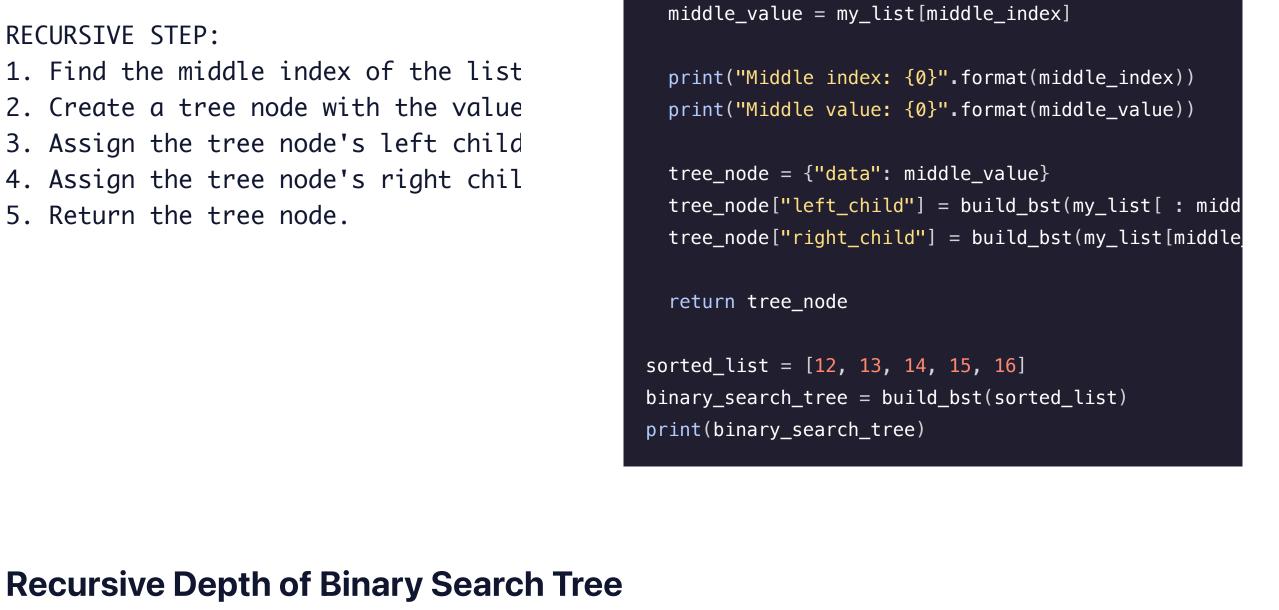
To build a binary search tree as a recursive

RECURSIVE STEP: 1. Find the middle index of the list 2. Create a tree node with the value

If the list is empty, return "No Chi

4. Assign the tree node's right chil 5. Return the tree node.

3. Assign the tree node's left child



builds a sorted input list into two subtrees. The left child of the subtree contains a value that is less than the root of the tree. The right child of the subtree contains a value that is greater than

the root of the tree. A recursive function can be written to determine the depth of this tree. **Sum Digits with Recursion**

A binary search tree is a data structure that

Summing the digits of a number can be done recursively. For example: 552 = 5 + 5 + 2 = 12

```
def sum_digits(n):
 if n <= 9:
```

return sum_digits(n // 10) + last_digit

left_depth = depth(tree["left_child"])

right_depth = depth(tree["right_child"])

return max(left_depth, right_depth) + 1

Palindrome in Recursion

A palindrome is a word that can be read the

same both ways - forward and backward. For

example, abba is a palindrome and abc is not.

The solution to determine if a word is a palindrome can be implemented as a recursive function. **Fibonacci Iterative Function**

def fibonacci(n): if n < 0: raise ValueError("Input 0 or greater only!")

fiblist.append(fiblist[i-1] + fiblist[i-2])

0, 1, 1, 2, 3, 5, 8, 13, ... A function to compute the value of an index in the Fibonacci sequence, fibonacci (index) can

A Fibonacci sequence is made up adding two

previous numbers beginning with 0 and 1. For

example:

be written as an iterative function. **Recursive Multiplication**

The multiplication of two numbers can be solved recursively as follows:

Base case: Check for any number that

def multiplication(num1, num2): if num1 == 0 or num2 == 0: return 0 return num1 + multiplication(num1, num2 - 1)

Recursive step: Return the first num

Iterative Function for Factorials To compute the factorial of a number, multiply

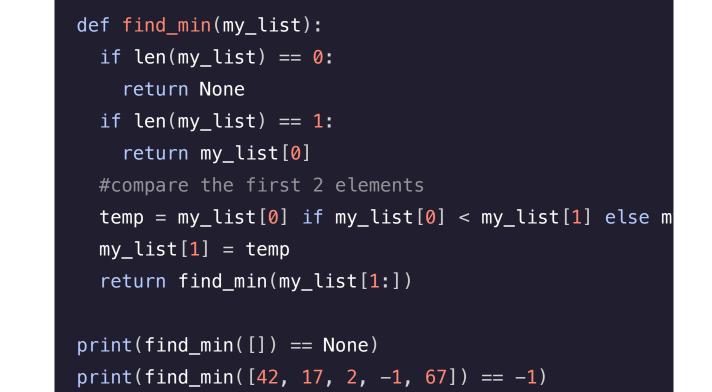
number. An example of an iterative function to compute a factorial is given below.

all the numbers sequentially from 1 to the

def factorial(n): answer = 1while n != 0: answer *= n n -= 1 return answer

Recursively Find Minimum in List

We can use recursion to find the element with the minimum value in a list, as shown in the code below.



← Previous

▶ Print Cheatsheet

≪ Share ▼

```
Learn Recursion with Python
Recursion gives you a new perspective on problem-
solving by defining a problem in terms of itself.
```

Plans

For individuals

For students

For teams

Discounts

Community

Code Crew

Clubs

Discord

Events

Forums

Learner Stories

Student Beans

	3 hours
Company	Resources
About	Articles
Careers	Blog
Affiliates	Cheatsheets
	Code challenges
🛚 f 🖸 🏭 🗗	Docs
	Projects
	Videos

Workspaces

Support

Help Center

Learn More on Codecademy

Course

With **Certificate**

code cademy

from skillsoft

Subjects	Languages	Career buildin
Al	Bash	Career paths
		•
Cloud Computing	С	Career center
Cloud Computing Code Foundations	C C++	•
	_	Career center
Code Foundations	C++	Career center Interview prep
Code Foundations Computer Science	C++ C#	Career center Interview prep Professional ce
Code Foundations Computer Science Cybersecurity	C++ C# Go	Career center Interview prep Professional ce